



Modeling Large Compute Nodes with Heterogeneous Memories with Cache-Aware Roofline Model

Nicolas Denoyelle, Brice Goglin, Aleksandar Ilic, Emmanuel Jeannot, Leonel Sousa

► To cite this version:

Nicolas Denoyelle, Brice Goglin, Aleksandar Ilic, Emmanuel Jeannot, Leonel Sousa. Modeling Large Compute Nodes with Heterogeneous Memories with Cache-Aware Roofline Model. High Performance Computing systems - Performance Modeling, Benchmarking, and Simulation - 8th International Workshop, PMBS 2017, Nov 2017, Denver (CO), United States. pp.91-113, 10.1007/978-3-319-72971-8_5 . hal-01622582

HAL Id: hal-01622582

<https://inria.hal.science/hal-01622582>

Submitted on 24 Oct 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Modeling Large Compute Nodes with Heterogeneous Memories with Cache-Aware Roofline Model

Nicolas Denoyelle^{1,3}, Brice Goglin¹, Aleksandar Ilic², Emmanuel Jeannot¹, and Leonel Sousa²

¹ Inria – Bordeaux - Sud-Ouest, Univ. Bordeaux, France

{nicolas.denoyelle, brice.goglin, emmanuel.jeannot}@inria.fr

² INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal

{aleksandar.ilic, leonel.sousa}@inesc-id.pt

³ Atos

Abstract. In order to fulfill modern applications needs, computing systems become more powerful, heterogeneous and complex. NUMA platforms and emerging high bandwidth memories offer new opportunities for performance improvements. However they also increase hardware and software complexity, thus making application performance analysis and optimization an even harder task. The Cache-Aware Roofline Model (CARM) is an insightful, yet simple model designed to address this issue. It provides feedback on potential applications bottlenecks and shows how far is the application performance from the achievable hardware upper-bounds. However, it does not encompass NUMA systems and next generation processors with heterogeneous memories. Yet, some application bottlenecks belong to those memory subsystems, and would benefit from the CARM insights. In this paper, we fill the missing requirements to scope recent large shared memory systems with the CARM. We provide the methodology to instantiate, and validate the model on a NUMA system as well as on the latest Xeon Phi processor equipped with configurable hybrid memory. Finally, we show the model ability to exhibits several bottlenecks of such systems, which were not supported by CARM.

1 Introduction

The increasing demands of current applications, both in terms of computation and amount of data, and the limited improvements of sequential performance of the cores led to the development of large multi-core and many-core systems [1]. These platforms embed complex memory hierarchies, spanning from registers, to private and shared caches, local main memory, and memory accessed remotely through the interconnection network. In these systems, memory throughput is not uniform anymore since the distance between processor and memory banks varies. On such Non-Uniform Memory Access (NUMA) architectures, the way data is allocated and accessed has a strong impact on performance [2]. Optimizing applications data locality for these machines requires a deep understanding of

hardware bottlenecks as well as application needs. Hence, modeling of memory access performance is of high importance.

Recently, the latest Intel Xeon Phi processor, codename Knights Landing (KNL) [3], entered the NUMA landscape with a processor divisible into 4 *Sub-NUMA Clusters* (SNC-4 mode). Usually, NUMA platforms include several sockets interconnected with processor-specific links (e.g. Quick Path Interconnect [4]) or by custom technologies such as SGI NUMalink or Bull Coherent Switch [5]. However, the KNL interconnects NUMA clusters at the chip scale (through a 2D mesh of up to 36 dual-core tiles). Though the software may see both types of system as similar homogeneous NUMA trees, the strong architectural differences between NUMA sockets and KNL chips, described above, can impact application performance in different ways and motivate the joint study of both systems.

Additionally, each cluster of the KNL may feature traditional DDR memory as well as 3D-stacked high-bandwidth memory named MCDRAM, that can be used as a hardware-managed cache or as an additional software-managed memory. Managing heterogeneous memories in runtime systems, applications or compilers brings another level of complexity and makes performance analysis harder and even more necessary. Hence, being able to understand the impact of the memory hierarchy and core layout on application performance as well as on attainable hardware upper-bounds is of high interest. This is especially true when modeling the architecture and tuning applications to this kind of hardware.

To optimize the application execution and to infer their ability to fully exploit the capabilities of those complex systems, it is necessary to model and acquire the knowledge about the realistically achievable performance upper-bounds of these systems and their components (including all levels of memory hierarchy and interconnection network). The Cache-Aware Roofline Model [6] (CARM) has been recently proposed (by some of the authors of this paper) as an insightful model and an associated methodology aimed at visually aiding the performance characterization and optimization of applications running on systems with cache memory subsystems. CARM has been integrated by Intel into their proprietary tools, and it is described as “*an incredibly useful diagnosis tool (that can guide the developers in the application optimization process), ensuring that they can squeeze the maximum performance out of their code with minimal time and effort.*”⁴ However, the CARM only refers to systems based on a single-socket computational node with uniform memory access, without considering the NUMA effects that can also dramatically impact the performance.

To address these issues, we propose a new methodology to enhance the CARM insightfulness and provide locality hints for application optimization on contemporary large shared memory systems, such as multi-socket NUMA systems and Many Integrated Core processors equipped with heterogeneous memory technologies and with various hardware configurations. The proposed model is experimentally validated with high accuracy on both an Intel Knights Landing and a dual-socket Broadwell Xeon multi-core host by relying on a set of micro-benchmarks, a set of synthetic benchmarks, and finally proxy applications.

⁴ Intel Advisor Roofline - 2017-05-12: <https://software.intel.com/en-us/articles/intel-advisor-roofline>

The remainder of this paper is organized as follows. Section 2 provides an in-depth overview of the Cache Aware Roofline Model and our contribution to make the model usable for NUMA and KNL architectures. Section 3 deep dives into the methodology to measure hardware upper-bounds for these systems. Sections 4 and 5 detail the model instantiation and validation for a Xeon E5-2650L v4 NUMA system composed of 4 NUMA nodes and the latest Xeon Phi many-core processor. Finally, Section 6 gives an overview of state-of-the-art related works.

2 Locality Aware Roofline Modeling

The generic Roofline modeling [7] is an insightful approach to represent the performance upper-bounds of a processor micro-architecture. Since computation and memory transfers can be simultaneously performed, this modeling is based on the assumption that the overall execution time can be limited either by the time to perform computations or by the time to transfer data. Hence, from the micro-architecture perspective, the overall performance can be limited either by the peak performance of computational units or by the capabilities of the memory system (*i.e.* bandwidth).

To model the performance limits of contemporary multi-core systems, the Cache-Aware Roofline Model (CARM) [6] explicitly considers both the throughput of computational unit and the realistically achievable bandwidth of each memory hierarchy level⁵. With this purpose, the CARM (see Figure 1) includes several lines representing the system upper-bounds (*Roofs*). Oblique lines (representing the memory bandwidths) cross the horizontal lines (representing the peak compute performance), bounding hereby the area of application characterization by respectively memory and compute regions. The CARM introduces a detailed and meticulous methodology for platform benchmarking from which this paper inherit and extends the content to NUMA platforms.

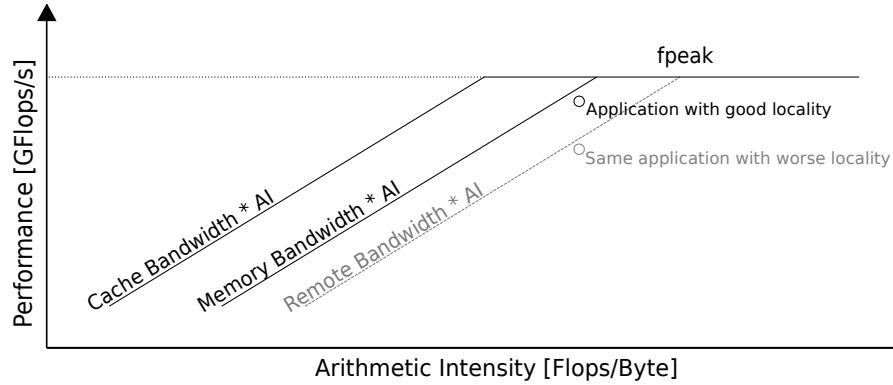


Fig. 1: CARM chart of an hypothetical compute node composed of one cache level and NUMA memories.

⁵ Main memory and cache levels.

In contrast to the other roofline approaches [7], the CARM perceives the computations and memory transfers from a consistent micro-architecture point of view, *i.e.* cores where the instructions are issued. Hence, when characterizing the applications, the CARM relies on the performance (in GFlop/s) and the true *Arithmetic Intensity* (AI), *i.e.* the ratio of performed compute operations (flops) over the total volume of requested data (in bytes). The CARM is presented in the log-log scale, where the x-axis refers to the AI (in flops/byte) and the y-axis refers to the performance (in GFlop/s).

Our Contribution: extending the CARM to NUMA and KNL

From the application perspective, the memory of modern computing systems is abstracted as a flat address space. However, the memory architecture of contemporary large compute nodes is made of remote and/or heterogeneous memories. In order to fully exploit those system capabilities, current software interfaces [8][9][10] require an explicit data allocation policy and/or thread binding policy to reach good performance [11][12]. Figure 2 depicts such a system, in-

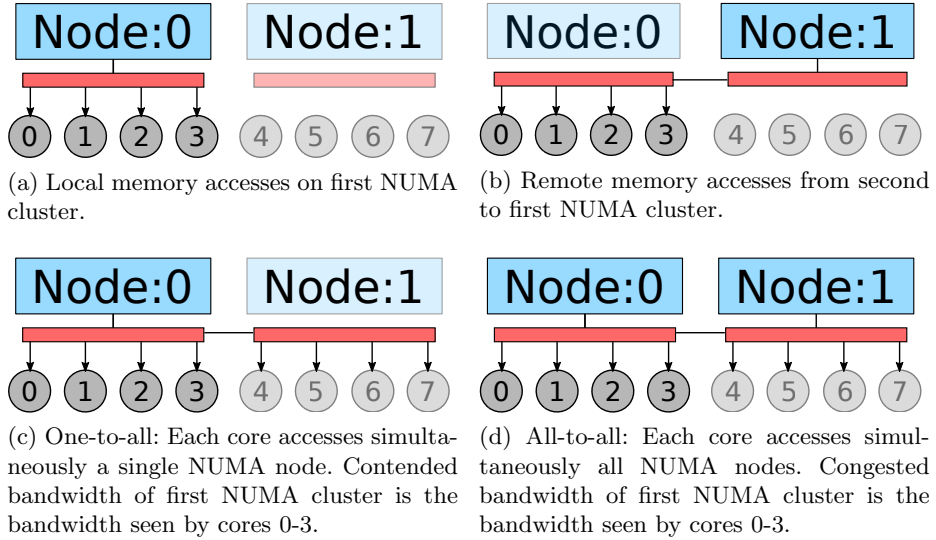


Fig. 2: Modeled memory access patterns.

cluding two sockets with their local memory (also named NUMA node) and a set of cores. On these systems, the bandwidth is not uniform across the network, and it influences memory access performance. Hence, when modeling, the source and destination of memory access (*i.e.* from a core to a NUMA node, e.g local access: Figures 2a or remote access: Figure 2b) should be taken into account to understand application performance. Moreover, such large scale systems contain a high amount of cores whose pressure on NUMA nodes can cause data accesses to be serialized when all of them are accessing a single memory. We qualify this

situation as *Contention* and depict it in Figure 2c. Finally, the network connecting the NUMA nodes to the cores can be subject to *Congestion*, when several data paths from the memory to the cores, cross the same link. In Figure 2d, we consider the case where data is balanced on memories, i.e there is no contention, however, each core will access data located over the whole system NUMA nodes and will eventually create Congestion because of the interleaved data paths. In the remainder of the paper, we use the term Cluster to refer to a set of neighbor cores and their local NUMA node(s)⁶.

CARM metrics are consistent across the whole memory hierarchy of a single cluster (as illustrated for the first cluster in Figure 2a). However, from the core perspective, memory access performance is not consistent across the system: bytes transferred from one NUMA node to a cluster are not transferred at the same speed to other clusters. This implies that the legacy CARM can only handle a single multi-core cluster, and fails to characterize accurately the cases in Figures 2b, 2c, and 2d. Yet, as Figure 1 shows, without proper (here remote) bandwidth representation in the CARM, locality issues are not obvious since the performance loss can come from many different sources: no vectorization, sparse memory access, *etc.*

Thus, we propose to extend the CARM with the Locality Aware Roofline Model (LARM), providing the lacking NUMA insights, represented in Figure 2 and characterizing the three main throughput bottlenecks, characteristic of this type of hardware: non uniform network bandwidth (Figure 2b), node contention (Figure 2c), and network congestion (Figure 2d). For this purpose, the LARM iterates the CARM over all the clusters of a computing system and keeps local consistency while minimizing the changes over the legacy model and taking into account the non-uniform aspect of the system. It follows that the LARM chart is a set of CARM charts, *i.e.* one chart per cluster, characterizing hereby the system performance upper-bounds under all perspectives. In each subsequent chart, the LARM includes three new groups of roof characterizing above mentioned bottlenecks. The remote roofs set the reference upper-bound of achievable bandwidth from remote nodes to a cluster. The congestion roof set the bandwidth achieved by a cluster when all the system cores are accessing simultaneously memory regions located across every NUMA nodes in a round-robin fashion. Finally the contention roofs characterize a cluster granted bandwidth when the whole system cores are accessing simultaneously a single NUMA node. Unlike usual CARM roofs, the new roofs stand as lower-bound roofs because they represent a reference below top expectations, *i.e.* local memory access roofs. However, as for the CARM, the closer an application is to a roof, the more likely this application is to be bound by this hardware bottleneck.

To the best of our knowledge, there is no work using the CARM to characterize NUMA platforms. Hence, beside the contribution of extending the CARM, we present the following work in the remainder of this paper: 1) we implemented a tool based on CARM methodology and the proposed improvements to automat-

⁶ On usual platforms, a cluster is identical to the widely-used definition of a NUMA node. On KNL, there can exist two local NUMA nodes near each core (DDR and MCDRAM), hence two NUMA nodes per cluster.

ically instantiate and validate the model on multi-socket systems and Knights Landing (KNL) Xeon Phi (for various memory configurations); 2) we thus validate the new model with high accuracy micro-benchmarks, for both systems; 3) We also demonstrate the model usability with synthetic benchmarks from the BLAS package; and 4) we exhibit the model ability to pinpoint data locality issues on MG from the NAS parallel benchmarks [13] and Lulesh proxy-application [14], where several data allocation policies are applied.

3 Methodology for Memory and Micro-Architecture Throughputs Evaluation

Initially, the Cache-Aware Roofline Model is built with two sets of parameters: micro-architecture instruction throughput and the attainable memory bandwidth. The former provides the peak floating point performance and L1 bandwidth while the latter is used to construct a set of local memory roofs (i.e L2, L3, local DRAM bandwidths). The Locality-Aware Roofline model, adds the perspective dimension and rooflines for several memory access patterns which require the ability to detect and model the system topology. For this purpose, we leverage hwloc [9] hierarchical representation of the machine to automatically enrich the CARM with the herein proposed memory roofs.

The micro-architecture throughput can be obtained whether by relying on the theoretical hardware properties, or by extensively benchmarking the micro architecture. In the former case, the peak floating point performance can be computed as:

$$\underbrace{F_{peak}}_{GFlop/s} = \underbrace{Throughput}_{Instructions/Cycle} * \frac{Flops}{Instruction} * N * \underbrace{Frequency}_{GHz}, \quad (1)$$

where the *Throughput* is the number floating point instruction retired per cycle by one core, *Flops/instruction* is the number of floating point operations performed in each instruction (e.g 2 for FMA instruction and 1 for ADD instruction), and *N* is the number of cores considered. Similarly, the peak bandwidth of the Level 1 cache can be computed as:

$$\underbrace{Bandwidth}_{GByte/s} = \underbrace{Throughput}_{Instructions/Cycle} * \frac{Bytes}{Instruction} * N * \underbrace{Frequency}_{GHz}. \quad (2)$$

Sometimes, theoretical throughput, provided by the constructor, and experimental throughput measured from highly tuned software do not match, or even the former is not publicly available. For this reason, we use the prior CARM methodology to implement highly optimized micro-benchmarks and build the proposed roofs. Our methodology for NUMA-specific bandwidth evaluation relies on a hierarchical description of the system topology as provided by the hwloc library, to characterize the system bandwidth in a pertinent way. We focus on deep and heterogeneous memory level evaluation, rather than on micro-architecture through-

put evaluation and caches already studied in [6]. Since the model needs to provide insights on possible bottlenecks of NUMA systems, the model includes the bandwidth roofs described in Section 2 and Figure 2, *i.e.* local accesses, remote accesses, accesses with congestion and accesses with contention. In order to characterize local and remote bandwidths of a cluster in the model, a benchmark performs contiguous memory access, as in the CARM, but on each NUMA node individually. One thread per core of the target cluster is spawned, then for each roof (*i.e.* local and remotes), the workload is iteratively allocated on each NUMA node, as depicted in Figures 2a and 2b. We do not look at individual links, but rather at pairs of cores+NUMA node, even though sometimes there are multiple (unknown) hops between clusters. The contended bandwidths are obtained similarly to the local and remote bandwidths, but loading the whole system cores with threads (Figure 2c). Each cluster granted bandwidth is associated with the source contended node to build the contended roofs on each cluster chart. Finally, the congested bandwidth is obtained by doing memory access from all the cores, contiguous on the virtual address space, but with pages physically allocated in a round-robin fashion across the system NUMA nodes, and with a private data set for each thread. Once again, the bandwidth perceived by each cluster is modeled as the congested roof in its local CARM. Though we call it congestion, it differs from the official definition⁷. However it fits a more practical and easy to reproduce memory access pattern, *i.e.* the one implied by using the linux interleaved memory allocation policy.

In this paper, we only show the bandwidth of `LOAD` instructions because it suits better our use cases, however we are able to also measure `STORE`, non-temporal `STORE` and mix of those for all memory levels with our tool.

4 Model Instantiation and Validation on Multi-Socket System

In order to set up and validate the model, we use a dual-socket NUMA system named *Joe0*. It is composed with two Broadwell Xeon E5-2650L v4 processors (at 1.7 GHz), configured with the cluster-on-die mode and exposing the 4 NUMA nodes to the system. Each NUMA node of the system topology (Figure 3) implement 7 cores, here with hyperthreading disabled, and is pictured on Figure 3.

4.1 Platform Evaluation and Model Instantiation

By relying on the testing methodology proposed in [6], it was possible to reach near theoretical compute and L1 cache throughputs on the Intel Broadwell micro-architecture, as presented in Table 1. Each core throughput is derived using the number of operations per instruction and the processor frequency to obtain the peak FMA floating point performance (reaching 190 GFlop/s for a single cluster).

⁷ Network congestion in data networking and queueing theory is the reduced quality of service that occurs when a network node is carrying more data than it can handle.

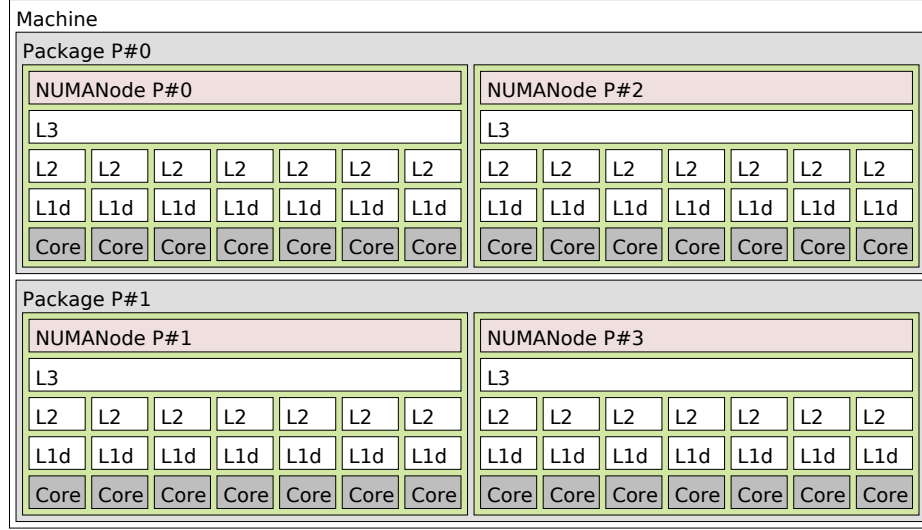


Fig. 3: hwloc topology representation of a dual-socket Xeon E5-2650L v4.

Instruction Throughput	Load	Store	ADD	MUL	FMA
Theoretical	2	1	1	2	2
Experimental	1.99	0.99	0.99	1.99	1.99

Table 1: Joe0 core instructions throughput (Instructions/Cycle)

Memory Level	Bandwidth (GByte/s)
L1	760.1
L2	309.2
L3	154.0
NUMANODE:0 (local)	36.1
NUMANODE:1 (remote)	17.5
NUMANODE:2 (remote)	15.0
NUMANODE:3 (remote)	14.3
NUMANODE:0 (contended)	16.7
NUMANODE:1 (contended)	8.3
NUMANODE:2 (contended)	6.8
NUMANODE:3 (contended)	6.2
All NUMANODES (congested)	18.1

Table 2: Joe0 bandwidth roofs to the first NUMA cluster, i.e NUMANODE:0.

As presented in Section 3, the next evaluation aims to extensively benchmark the memory subsystem with several memory access patterns, *i.e.* the remote/local bandwidth between each pair (cluster, NUMA node), as well as the contended/-congested ones. The results obtained are presented in Table 2 for the first NUMA cluster of the system. Unless specified, the model presented herein is restricted to a single NUMA cluster, due to the bandwidth symmetry between clusters.

The obtained measures (Tables 1, 2) are then used to build the proposed model depicted in Figure 4 for the first cluster of Joe0. Besides the roofs for local caches, this CARM chart also includes all the proposed memory roofs, namely local, remote, contended and congested roofs.

4.2 Model Validation

With Micro-Benchmarks This validation step consists in micro-benchmarking the system with several arithmetic intensities, *i.e.* interleaving the memory and compute instructions used in the above platform evaluation. It assesses code ability to reach measured roofs while performing both computations and memory accesses. We measure the roofs fitness as the relative root mean squared error⁸ of validation points to the roof performance for a realistic range of arithmetic intensities. The errors and deviation (too small to be visible) for each validation point, and for each bandwidth roof of a single cluster of the system are presented in Figure 4. As the error computed in the legend is small (less than 2% in average for every roof), the validation enforces that measured bandwidths are attainable by programs of various arithmetic intensities.

With Synthetic Benchmarks Figure 5 shows the LARM instantiated on the first socket of Joe0. For each NUMA cluster a CARM chart includes local cache bandwidths, local node bandwidth, the bandwidth under congestion⁹ and the bandwidth of the first NUMA node under contention (which is different whether we see it from the first or the second cluster). Figure 5 also illustrates the memory-bound ddot kernel and the compute-bound dgemm kernel from the BLAS package, under several scenarios, showing the model ability to pinpoint locality issues. For each scenario, threads are bound in round-robin fashion and data allocation policy is one of: firsttouch (*i.e.* data in memory close to threads), interleave (*i.e.* data spread on all nodes), Node:0 (*i.e.* data on a single memory node). Each thread performs the same amount of work though the allocation policy on a single node may create an asymmetry when observing their performance across different NUMA nodes (Figure 5). The modeled applications were run on the full system, *i.e.* 28 threads (1 thread per core), however, only the model for a single socket is presented to avoid redundancy. On the chart (Figure 5), ddot and dgemm are represented each with their own constant arithmetic intensity (*i.e.* the

⁸ The error is computed as $\frac{100}{n} \times \sqrt{\sum_{i=1..n} \left(\frac{y_i - \hat{y}_i}{\hat{y}_i} \right)^2}$ where y_i is the validation point at a given arithmetic intensity, and \hat{y}_i is the corresponding roof.

⁹ Remote memory bandwidths are very close to congested bandwidths on this system and we omit the former in the chart to avoid confusion.

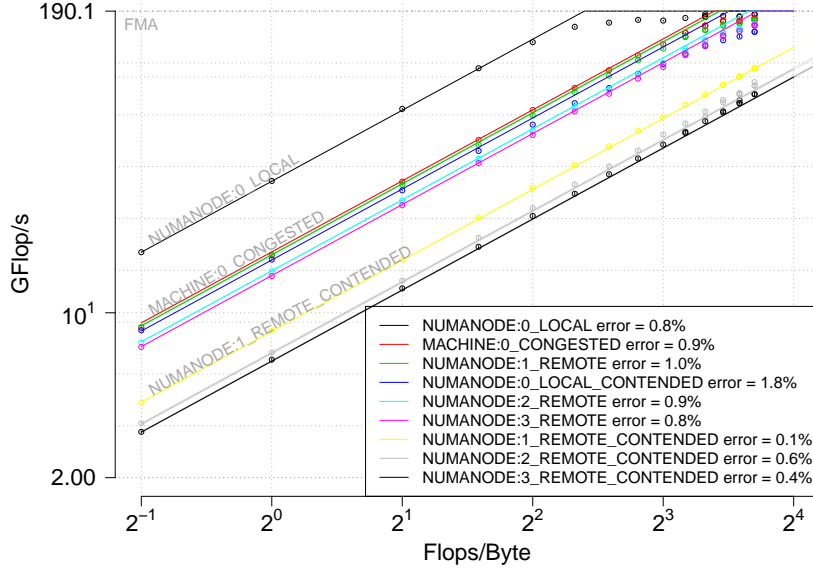
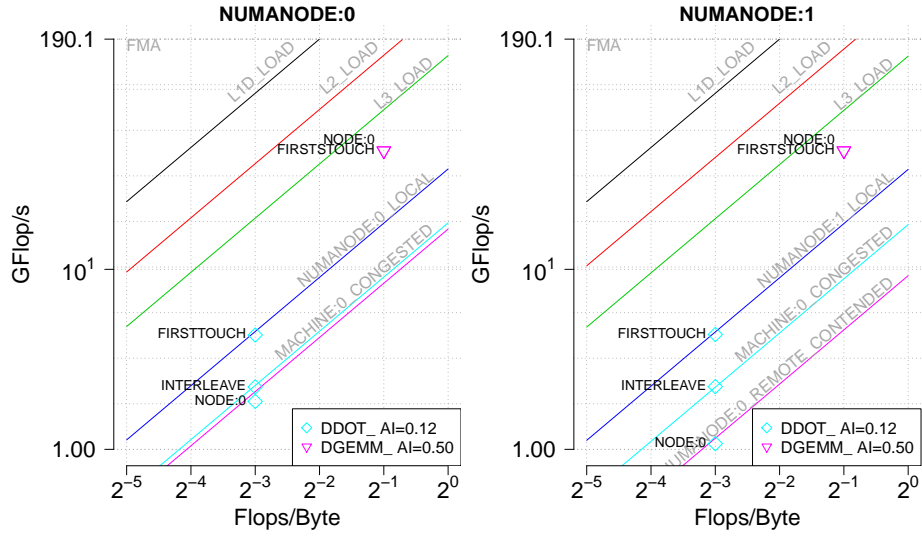


Fig. 4: CARM validation of one NUMA cluster of Joe0 platform. Validation points are visible along the roofs. Finally the model error for each roof is in the legend.



(a) CARM of the first NUMA cluster. (b) CARM of the second NUMA cluster.

Fig. 5: LARM chart of linear algebra kernels on one socket of Joe0 system.

code is unchanged between scenarios) but with several performances (changed runtime parameters).

The ddot case with allocation on Node:0 has a different performance whether we look at the first or the second cluster. Hence, the kernel characterization shows the model ability to spot asymmetries. Even if asymmetries do not originate from the instructions, they can come from the data distribution and significantly impact the performance [15]. Congested and contended roofs also successfully characterize similar bottlenecks in ddot application. Indeed, in Figure 5, ddot kernel with interleaved access (*i.e.* inducing congestion) and access on a single node (*i.e.* inducing contention) match with appropriate memory bandwidths. Optimized compute-intensive applications do not suffer from locality issues. Indeed, as presented in Figure 5, the dgemm execution is not affected by non-uniform memory access, since it achieves the same performance on each node even if data is allocated with different policies. This can be attributed the high cache efficiency of the kernel allowing the system to prefetch the required data into the cache before it is actually required, thus avoiding the local and remote memory access bottlenecks.

In a nutshell, data allocation policies applied to synthetic benchmarks affects the performance in a way that is foreseeable. It matches expectations from their characterization in the LARM, thus validating the proposed roofs relevance.

With NAS MG Parallel Benchmark This step aims to show that the model insights can help to flush out performance bottlenecks, i.e application characterization relatively to the new roofs can help to pinpoint potential execution bottlenecks. For this purpose, we ran a C version¹⁰ of the NAS-3.0 MG benchmark with one thread per core, bound in a round-robin fashion on the system cores. On this system, we extract the LARM metrics with hardware counters at the core level, and aggregate the results at the Cluster level. As presented in Figure 6, three functions from MG benchmark are characterized on the first cluster of the system with several memory allocation strategies. In the first scenario, the default linux policy *firsttouch* is used for data allocation. The characterization of these functions (labeled with *firsttouch*) reach near contention roof performance, and suggest to use the *interleave* allocation policy to balance memory accesses over the NUMA nodes in order to decrease the contention. Indeed, the latter policy increases dramatically the performance above the congestion roof. However, it is unlikely that the *interleave* policy surpasses the *firsttouch* policy with such a significance. Hence this observation also suggests that *firsttouch* actually allocates memory on a single node. Indeed, once parallelized, the previously sequential memory allocations, enable the *firsttouch* policy to allocate data on all NUMA nodes near appropriate threads, and improve again the performance (labeled with *enhanced_firsttouch*) compared to the *interleave* policy.

To sum up, the LARM characterization of the memory-bound MG benchmark, matches the contended roofs when data-allocation is serialized, i.e data is allocated on a single contended node, and improves above the congestion roof once the contention issue is solved, validating hereby the proposed roofs.

¹⁰ <https://github.com/benchmark-subsetting/NPB3.0-omp-C>

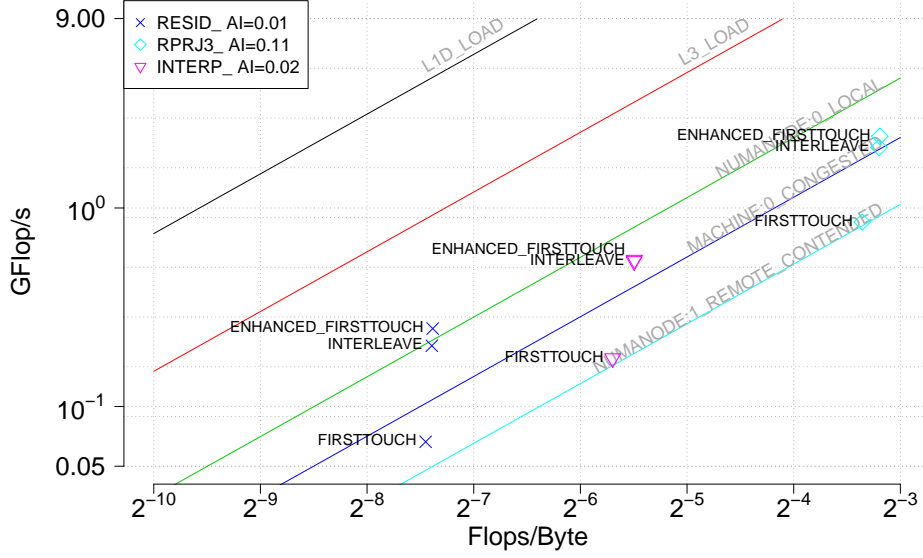


Fig. 6: NAS-3.0 MG Functions Characterization on Joe0 first cluster.

5 Model Instantiation and Validation on Knights Landing Processor

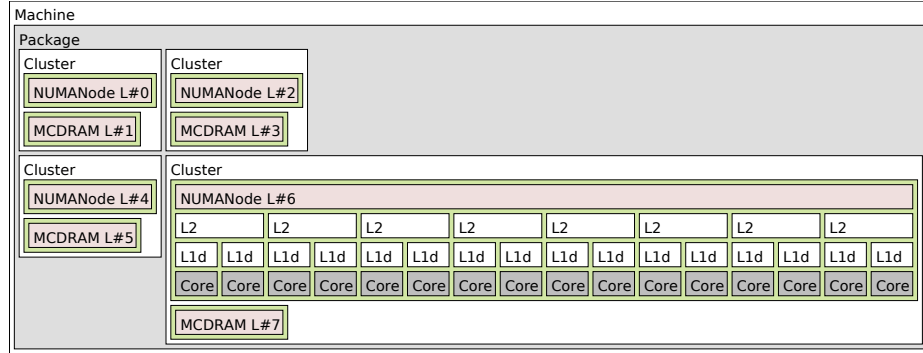


Fig. 7: hwloc model of KNL topology in SNC-4 flat mode. Only the fourth cluster is detailed, for clarity. Other clusters have a similar topology.

When in SNC-4 mode [3], the KNL is a special case of a multi-Socket system where each socket has an additional fast memory (MCDRAM) to the conventional memory (DRAM also specified as NUMA:i), which is addressable in *Flat* mode or configurable as a last level cache in *Cache* mode. Whether the flat mode or the cache mode is used, the system may yield different bandwidths, performance and execution time, and thus, the proposed model changes accordingly.

For our experiments, we used Knights Landing 7230 chips, with 64 cores at 1.3GHz. The topology of the KNL with SNC-4 flat configuration is shown in Figure 7 where the complete topology is provided for the last cluster. The mesh interconnection network (Figure 8) between L2 tiles of the chip is widely different from conventional multi-socket system [16] and motivates additional observations compared to the previous system.

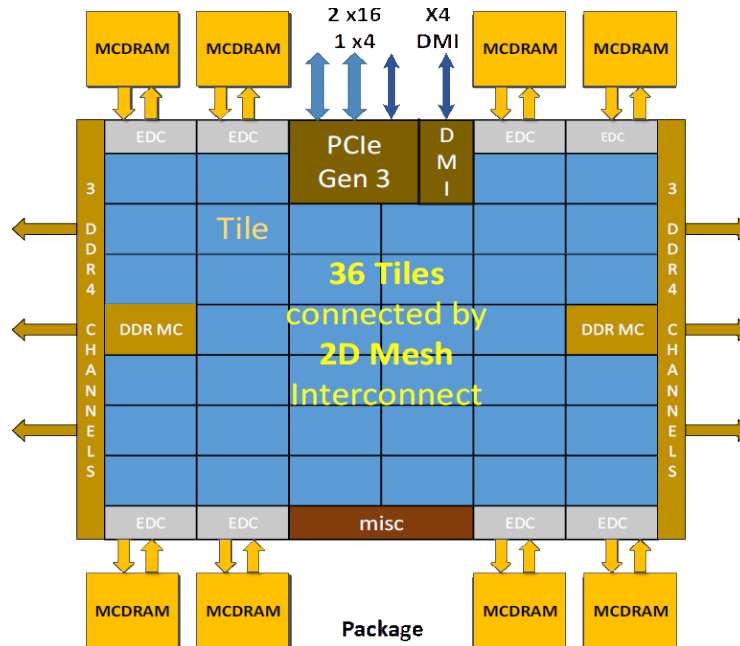


Fig. 8: Architecture of the Knights Landing mesh interconnect with DRAM and MCDRAM memory controllers (Source: Intel). Only 32 of these 38 tiles are actually enabled in our experimentation platform. The amount of tiles enabled can reach up to 36 tiles, though 38 are present.

5.1 Platform Evaluation and Model Instantiation

By relying on the micro-architecture evaluation methodology from Section 3, the highest achievable throughput with carefully design micro-benchmarks is slightly lower than the theoretical values (see Table 3). However a performance of 2.2 TFlop/s for 64 cores is still achieved.

Table 4 presents the bandwidth evaluation between clusters solo (*i.e.* by fully exercising memory units with a single cluster) for the flat mode. Only the evaluation for the first two clusters is presented since the others yield a similar bandwidth. Contrary to the multi-socket system, remote and local DRAM attain similar bandwidths which suggests high efficiency of the KNL interconnection network. However, significant and less predictable variations can be noticed for

Instruction	Load	Store	ADD	MUL	FMA
Theoretical throughput	2	1	2	2	2
Experimental throughput	1.66	0.96	1.70	1.70	1.70

Table 3: Theoretical and experimental instruction throughput (in instructions per cycle) for a single core of the KNL platform.

MCDRAM, which would require the disclosure of more architectural details to fully explain the mesh behavior.

		from			
		NUMA:0	MCDRAM:1	NUMA:2	MCDRAM:3
to	Cluster:0	38.1±0.1	92.0±0.5	38.0±0.8	86.6±0.4
	Cluster:1	38.1±0.1	91.5±0.4	38.2±0.1	92.8±0.4
	Cluster:2	37.8±0.1	90.6±0.5	38.1±0.1	83.7±0.6
	Cluster:3	38.0±0.2	82.8±0.4	38.0±0.1	90.8±0.3

Table 4: KNL load bandwidth (GByte/s) from first and second clusters memories to cores in flat mode. Other clusters are omitted because of similar results.

In Table 5, we also compare the load bandwidth granted to the first cluster when the data set is allocated into the first cluster DRAM and MCDRAM under several scenarios. The very first line is the reference when the cluster runs solo as in Table 4 but comparing the cache and flat modes. In the cache mode, the bandwidth of both types of memories (*i.e.* DRAM and MCDRAM) decreases, probably due to the overheads induced by the MCDRAM caching mechanism. In both modes, the DRAM bandwidth (NUMA:0) reduces when using all clusters simultaneously (*i.e.* local with 64 versus 16 threads), whereas this is less obvious for the MCDRAM. The presence of only two DRAM memory controllers shared among 4 clusters to access DRAM memory, whereas there are 8 EDC controllers (two per cluster) to access the MCDRAM (see Figure 8), is a possible cause of this behavior.

		flat		cache		
		NUMA:0	MCDRAM:1	NUMA:0	MCDRAM:1	threads
Cluster:0	local	38.1 ±0.1	92.0 ±0.5	22.9 ±0.7	85.4 ±3.0	16
	local	21.7 ±0.7	90.9 ±1.2	20.0 ±0.7	83.3 ±2.0	64
	congested	19.8 ±0.3	77.6 ±2.0	17.0 ±0.4	NA	64
	contended	10.7 ±0.0	21.5 ±0.5	NA	NA	64

Table 5: KNL load bandwidth (GByte/s) from first cluster memories.

In the cache mode, the bandwidth drop of DRAM memory when using all clusters simultaneously is not as high as the drop in the flat mode, probably because of data reuse in MCDRAM cache, which redirects a part of the traffic via the EDC

channels and absorbs a part of the contention on DRAM memory controllers. Congestion already happens for interleaved memory access on DRAMs, provoking a further bandwidth reduction when compared to local memory accesses. As expected, contention is the worse case scenario, resulting in a dramatic bandwidth reduction for the cluster. Since congestion and contention are observable, they imply a need for locality to get good performance. Several Non-Achievable values stand in Table 4. One of them, i.e contention on NUMA:0 in cache mode cannot be observed with the technique used in Section 3 methodology. Indeed, the private data accessed by each cluster in NUMA:0 memory would actually fit into the 4 MCDRAMs and result in MCDRAMs benchmark instead of NUMA:0 benchmark.

Based on the above characterization, the LARM is constructed for a single cluster and presented in Figure 9, where the chip is configured in (SNC-4), flat mode. In contrast to the previous platform, it also includes the MCDRAM roofs siblings of the DRAM roofs. Bandwidths of remote nodes are hidden for clarity because they have the same order of magnitude as the local bandwidth and thus overlap on the chart.

5.2 Model Validation

With Micro-Benchmarks We use again the previous methodology to validate the model in Figure 9, for one cluster (equivalent on the others). The micro-benchmark validation on KNL fits the model with an error below 5% in average for each roofs. Most of it is due to the points located near the ridge on L1 and MCDRAM bandwidths roofs. Otherwise it still fits nearly perfectly the roofs in the memory-bound and in the compute-bound regions.

With Synthetic Benchmarks As previously referred, the validation with synthetic benchmarks aims to verify that well chosen causes lead to expected consequences, i.e well chosen synthetic benchmarks are able to hit the roofs. For this purpose, we characterize again `ddot` and `dgemm` BLAS kernels in the CARM chart (Figure 10) of the first cluster of the KNL. Here we focus on MCDRAM usage rather than the classical memory allocation policy already studied for the multi-socket system. Hence, we compare several data allocation strategies and sizes, as well as the flat and cache configurations of the chip.

The `ddot` function is compared under both flat and cache modes and by adopting various allocation strategies and data-set sizes. The small data-set (labeled *small* in Figure 10) fits into the MCDRAM, whereas the large data-set (labeled *large*) does not. In the MCDRAM policy, we allocate the whole *small* data-set into MCDRAM. In the interleave policy, we use the Linux policy allocating pages of the data-set across all the nodes, *i.e.* MCDRAM and DRAM nodes. Finally, the Linux policy `firsttouch` allocates data on the DRAM near the first thread writing the corresponding page.

In flat mode, allocation into MCDRAM allows for the performance to reach near MCDRAM roof, and is visually assessed by the model. Unlike allocations

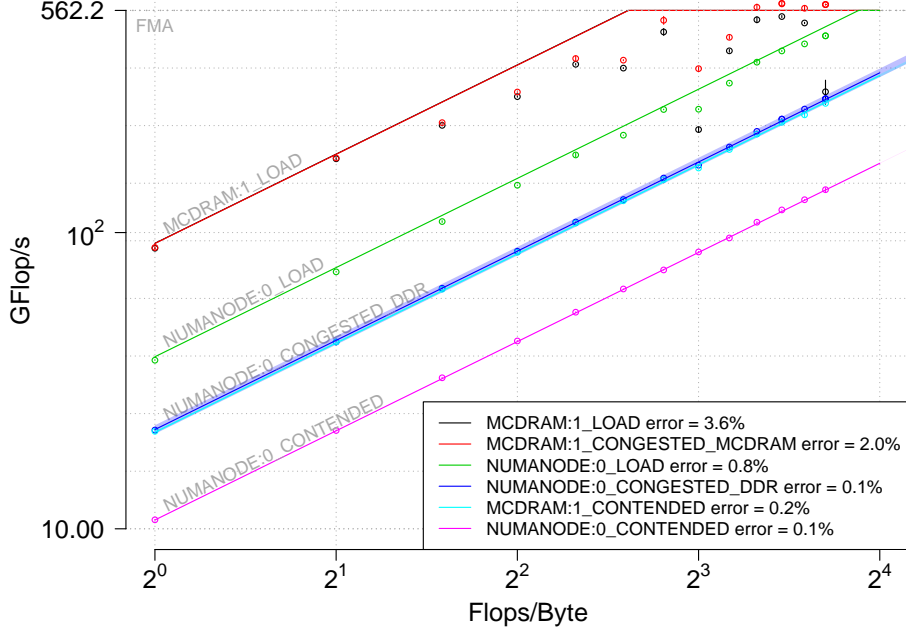
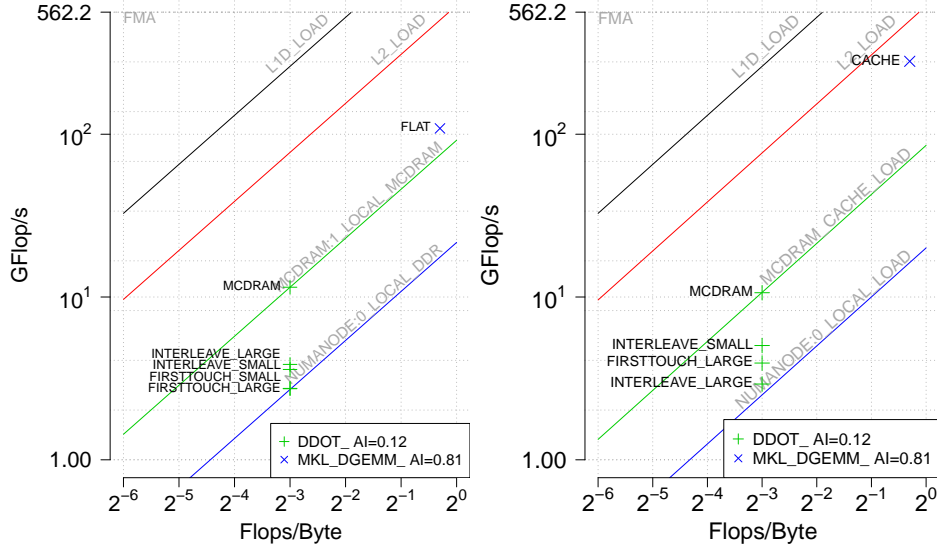


Fig. 9: CARM validation of one sub-NUMA cluster of KNL platform in SNC-4 flat mode.

into MCDRAM, large data-set with firsttouch policy are allocated into slow memory and also reach near DRAM roof performance. With the interleave policy, the data-set is mixed across memories and thus the performance stands in between the local DRAM roof and the local MCDRAM roof, with higher influence of the slower DRAM memory (the point is closer to this roof). In cache mode, the small data-set is cached in the MCDRAM, thus reaches a performance near MCDRAM bandwidth roof. Although the MCDRAM bandwidth is lower in cache mode, the performance of large data-set with firsttouch policy is higher than the DRAM roof. This is because of the large MCDRAM cache size that allows reusing a significant part of the data, thus improving the achievable performance. Interleaving memory accesses decreases the performance when compared to firsttouch policy, because of the congestion hereby created.

The dgemm function uses a data set too large to fit into MCDRAM and is compared in flat or cache mode. As expected, the intrinsic temporal locality of this kernel allows the cache mode to yield a better performance. When choosing the dataset size or location (DRAM or MCDRAM) the synthetic benchmark performance still correspond to our expectation and validate the model insights on KNL system. When choosing the system configuration (flat or cache), the model also shows that kernels with good data reuse, i.e with a performance over MCDRAM roofs, benefit from the hardware cache mechanism and also validates the model relevance.



(a) CARM of the first cluster in flat mode. (b) CARM of the first cluster in cache mode.

Fig. 10: CARM of KNL first cluster with synthetic benchmarks running whether in flat mode or in cache mode.

With Lulesh proxy-applications For this NUMA system, we focus on the Lulesh application because of its sensitivity to memory bandwidth. The aim of this validation step, is to show whether the model helps managing memory, i.e. whether performance can be improved with the chip configuration or a good memory allocation policy. From Lulesh, we pick the three greatest memory-bound hot spots of the application (*i.e.* the ones bounded below NUMANode:0 roof), namely CalcFBHourGlassForElems, IntegrateStressForElems functions, and a loop in the main function. Due to the lack of required hardware counters, arithmetic intensity, performance and application profile are collected with the Intel advisor tool¹¹. In our experiments, we ran the application using a working set size large enough¹² not to fit into the MCDRAM. Hence, target memory needs to be carefully chosen to fulfill size constraints and a special care needs to be addressed when managing memory allocation to get good performance.

The first run allocates all data into regular DRAM memory (labeled DRAM), and aims at characterizing the application to find the potential allocation improvements. We then customize dynamic allocations in those hot spots by replacing the usual allocator from the standard C library with memkind [17] allocator to target the fast memory (labeled as *MCDRAM* in Figure 11a) instead of the

¹¹ Product version: Update 2 (build 501009).

¹² Lulesh application run parameters: -i 1000 -s 60 -r 4. The application is compiled with ICC 17.0.2 and options: -DUSE_MPI=0 -qopenmp -O3 -xHost

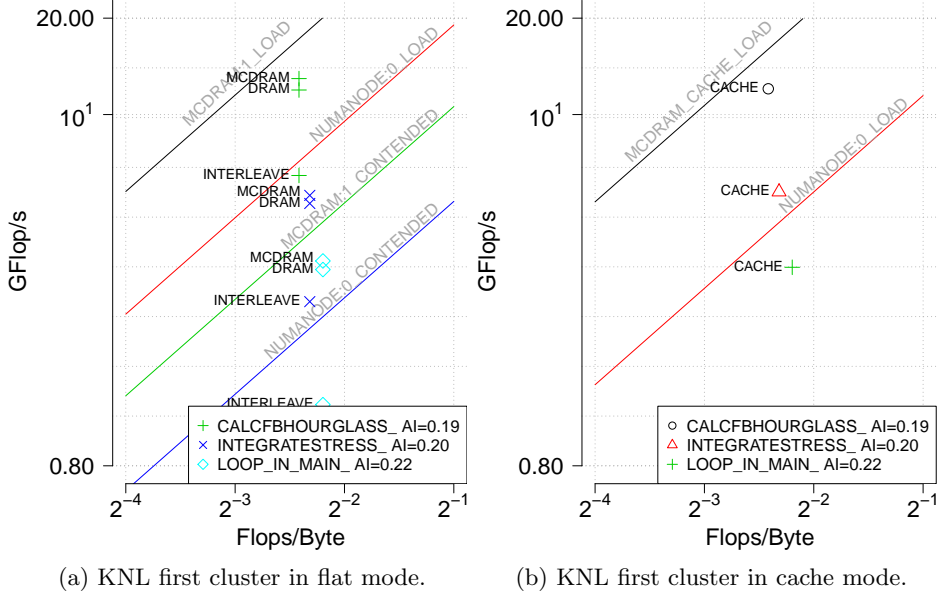


Fig. 11: CARM chart of the KNL first cluster. The 3 main hot spots as detected by Intel Advisor are represented both in cache and flat modes.

traditional DRAM (labeled as *DRAM*). Finally, instead of forcing MCDRAM allocations, we let the interleave policy (labeled as *interleave*) to choose data to put into MCDRAM for all allocations visible in the file `lulesh.cc`. Summarized, each hereby found hot spot is executed using three different policies, i.e DRAM allocation (labeled DRAM), custom allocations (labeled MCDRAM), and interleave policy, in flat mode (see Figure 11a).

The second chart in cache mode (see Figure 11b), contrasts the performance of hand-tuned allocations into the MCDRAM with the hardware management of the fast MCDRAM cache. As expected MCDRAM allocations provide higher performance in flat mode. However, in cache mode the hot spots characterization reaches comparable performance to the top achieved performances in flat mode, denoting the hardware efficiency to manage data locality. In comparison, the interleave memory policy performs poorly maybe because of the spread allocations forcing threads to access remote nodes, and congesting the mesh.

To summarize the use of the LARM for data allocation policy choice with `lulesh` on the KNL, the cache mode brings no significant performance improvement, and the characterization laying below the MCDRAM roofs gave us this insight. The data allocation policy however changes the performance dramatically between the interleave policy and other policies, but the cause is not clear in the model since the *interleave* points are widely spread on the performance axis which suggest another issue than ones the expected.

6 Related Works

To this date, there are two main approaches for Roofline modeling, namely: the Original Roofline Model (ORM) [7] and the Cache-Aware Roofline Model (CARM) [6]. Unlike the CARM that includes the complete memory hierarchy in a single plot, the ORM mainly considers the memory transfers between the last level cache and the DRAM, thus it provides fundamentally different perspective and insights when characterizing and optimizing applications [18]. Recently, the ORM was also instantiated on the KNL [19], without modifying the original model. The arithmetic intensity (AI) described in ORM is not to be confused with CARM AI because of the difference in the way how the memory traffic is observed. The bandwidth measured also differs from the one measured in this paper, the latter being explicitly load bandwidth. In [19], the authors present several ORM-based optimization case studies, and compare the performance improvements between Haswell processor and KNL, with data in DDR4 memory or MCDRAM, and finally KNL with data in MCDRAM memory. However, the authors do not show how the model can help choosing between memories when working sets do not fit in the fastest one nor they provide a comparison with the cache mode.

An extension to the ORM, named 3DyRM [20], has been proposed to provide locality insights on NUMA systems. This model considers memory accesses from a single last level cache to any other memory, and not only local memory. It extends the ORM with a latency dimension to characterize the sampled memory access. Not only 3DyRM inherits the distorted perspective of the ORM, when characterizing real-world applications, but also it gives very limited insights on the distance of memory accesses to the NUMA thresholds considered in this paper. Moreover, 3DyRM characterizes applications with sampled memory accesses, without classifying them nor providing a methodology to get the first order insights, which is the main goal of the legacy model.

Capability Model [16] is recently proposed to evaluate KNL realistic upper-bounds and guide applications performance optimizations. The authors established a complex model mostly focusing on latency and bandwidth of the mesh interconnect. The Capability Model focuses on communication intensive algorithms (such as barrier synchronization, reduction, sorting, *etc*), whereas the LARM has a throughput oriented approach, focusing on computational workloads stressing both compute and memory units. As such, the Capability models suits better message passing programming paradigms to enhance communication based algorithms, while the LARM suits better shared memory programming paradigms where communications are not explicitly expressed and mixed with computations.

Execution Cache Memory (ECM) [21] is also another insightful approach to model performance of memory-bound applications. This model is built under the similar assumptions as the CARM when modeling the performance of processing elements and memory levels, e.g., by considering their maximum throughput. However, the ECM aims at predicting the application runtime whereas the CARM aims at providing insights toward application characterization and optimization. Moreover, to the best of our knowledge, there are no studies demonstrating the

usability of the ECM for NUMA and heterogeneous memory systems featuring emerging heterogeneous memory technologies.

Our contribution to the CARM also advances its current implementation in the Intel proprietary tool's, referred as Intel Advisor Roofline [22], and for which some author of this paper published concrete cases usage [23]. Unlike Intel Advisor Roofline, we keep track of the MCDRAM bandwidth in several aspects, and provide additional insights about potential bottlenecks and characteristics of NUMA systems. Indeed, we demonstrated that our model improvements can efficiently spot locality related issues, and provide more insights, especially in the case of traditional multi-socket systems.

7 Conclusions

The trend of increasing the number of cores on-chip is enlarging the gap between compute power and memory performance. This issue leads to design systems with heterogeneous memories, creating new challenges for data locality. Before the release of those memory architectures, the Cache-Aware Roofline Model offered an insightful model and methodology to improve application performance with knowledge of the cache memory subsystem. With the help of hwloc library, we are able to leverage the machine topology to extend the CARM for modeling NUMA and heterogeneous memory systems, by evaluating the memory bandwidths between all combinations of cores and NUMA nodes.

Our contribution scopes most contemporary types of large compute nodes and characterizes three bottlenecks typical of those systems, namely contention, congestion and remote access. We showed that this additional information can help to successfully spot locality issues coming from parameters such as data allocation policy or memory configuration. To do so, we emphasized on several validation stages, ranging from micro-benchmarks to real-world applications on both a dual-Broadwell Xeon host and on an Intel Knight Landing processor. The LARM extension remains consistent with the traditional Cache-Aware Roofline Model while including a minimum of changes to the original methodology.

In the future we intend to validate the model also on larger systems embedding tens of NUMA nodes and probably yielding even more interest for locality aware modeling. It would also be interesting to investigate an extension of the model over the network in order to include distributed workloads characterization. Also, as mentioned in Section 3 footnote, we only consider the load bandwidth in the paper. However most applications mix load and store instructions and the top achievable roof in that case is neither the load bandwidth nor the store bandwidth but rather a combination of those. Additional constraints could also be added to the load/store mix in order to define a roof, but this would deserve a paper on its own. It could end up with an automatic roof matching features, which as for now, is left to the user.

Acknowledgments

We would like to acknowledge COST Action IC1305 (NESUS) and Atos for funding parts of this work, as well as national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

Some experiments presented in this paper were carried out using the PLAFRIM experimental testbed, being developed under the Inria PlaFRIM development action with support from Bordeaux INP, LaBRI and IMB and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux and CNRS (and ANR in accordance to the programme d'investissements d'Avenir, see <https://www.plafrim.fr/>).

References

1. Blake, G., Dreslinski, R.G., Mudge, T.: A survey of multicore processors. *IEEE Signal Processing Magazine* **26**(6) (November 2009) 26–37
2. Blagodurov, S., Zhuravlev, S., Dashti, M., Fedorova, A.: A Case for NUMA-aware Contention Management on Multicore Systems. In: 2011 USENIX Annual Technical Conference, Portland, OR, USA, June 15–17, 2011. (2011)
3. Reinders, J., Jeffers, J., Sodani, A.: Intel Xeon Phi Processor High Performance Programming Knights Landing Edition (2016)
4. Ziakas, D., Baum, A., Maddox, R.A., Safranek, R.J.: Intel® QuickPath Interconnect Architectural Features Supporting Scalable System Architectures. In: High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on, IEEE (2010) 1–6
5. : Bull atos technologies: Bull coherent switch. <http://support.bull.com/ols/product/platforms/hw-extremcomp/hw-bullx-sup-node/BCS/index.htm>
6. Ilic, A., Pratas, F., Sousa, L.: Cache-aware Roofline model: Upgrading the loft. *IEEE Computer Architecture Letters* **13**(1) (2014) 21–24
7. Williams, S., Waterman, A., Patterson, D.: Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* **52**(4) (April 2009) 65–76
8. Cantalupo, C., Venkatesan, V., Hammond, J., Czurylo, K., Hammond, S.D.: memkind: An Extensible Heap Memory Manager for Heterogeneous Memory Platforms and Mixed Memory Policies. Technical report, Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States) (2015)
9. Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., Namyst, R.: hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In IEEE, ed.: PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing, Pisa, Italy (February 2010)
10. Kleen, A.: A NUMA API for LINUX. Novel Inc (2005)
11. Lepers, B., Quema, V., Fedorova, A.: Thread and Memory Placement on NUMA Systems: Asymmetry Matters. In: 2015 USENIX Annual Technical Conference (USENIX ATC 15), Santa Clara, CA, USENIX Association (July 2015) 277–289
12. Chou, C., Jaleel, A., Qureshi, M.K.: CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache. In: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-47, Washington, DC, USA, IEEE Computer Society (2014) 1–12

13. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Simon, H.D., Venkatakrisnan, V., Weeratunga, S.K.: The nas parallel benchmarks. Technical report, The International Journal of Supercomputer Applications (1991)
14. Karlin, I., Keasler, J., Neely, R.: Lulesh 2.0 updates and changes. Technical Report LLNL-TR-641973 (August 2013)
15. Lepers, B., Quéma, V., Fedorova, A.: Thread and Memory Placement on NUMA Systems: Asymmetry Matters. In: USENIX Annual Technical Conference. (2015) 277–289
16. Ramos, S., Hoefer, T.: Capability Models for Manycore Memory Systems: A Case-Study with Xeon Phi KNL
17. : The Memkind Library. <http://memkind.github.io/memkind>
18. Ilic, A., Pratas, F., Sousa, L.: Beyond the Roofline: Cache-Aware Power and Energy-Efficiency Modeling for Multi-Cores. *IEEE Transactions on Computers* **66**(1) (Jan 2017) 52–58
19. Doerfler, D., Deslippe, J., Williams, S., Olike, L., Cook, B., Kurth, T., Lobet, M., Malas, T., Vay, J.L., Vincenti, H.: Applying the Roofline Performance Model to the Intel Xeon Phi Knights Landing Processor. In: International Conference on High Performance Computing, Springer (2016) 339–353
20. Lorenzo, O.G., Pena, T.F., Cabaleiro, J.C., Pichel, J.C., Rivera, F.F.: Using an extended Roofline Model to understand data and thread affinities on NUMA systems. *Annals of Multicore and GPU Programming* **1**(1) (2014) 56–67
21. Hofmann, J., Eitzinger, J., Fey, D.: Execution-Cache-Memory Performance Model: Introduction and Validation. *CoRR* **abs/1509.03118** (2015)
22. Intel: Intel Advisor Roofline (2017)
23. Marques, D., Duarte, H., Ilic, A., Sousa, L., Belenov, R., Thierry, P., Matveev, Z.A.: Performance analysis with cache-aware roofline model in intel advisor. In: 2017 International Conference on High Performance Computing Simulation (HPCS). (July 2017) 898–907